1. In this question we ask you to create and use a function which generates a sparse triadiagonal stiffness matrix for a system of \( n \) springs and masses connected in series. We show in Figure 1 the spring-mass system for the particular case \( n = 3 \).

![Spring-mass system](image)

Figure 1: Our spring-mass system for the particular case \( n = 3 \). Here the \( k_i, 1 \leq i \leq n \), are the values of the spring constants; note the values of the masses, \( m_i, 1 \leq i \leq n \), the applied forces, \( f_i, 1 \leq i \leq n \), and the displacements, \( u_i, 1 \leq i \leq n \), are not relevant in this set of exercises.

As described in class, the equilibrium displacement \( u \) may be found from the matrix equation \( K u = f \) where the \( i \)th row represents a force balance on the \( i \)th mass. The \( n \times n \) matrix \( K \) (MATLAB \( K \)) is denoted the stiffness matrix.

(a) The function

```matlab
def function [ K ] = generate_K( n, kvec )

K = spalloc(n,n,3*n);

K(1,1) = kvec(1) + kvec(2);
K(1,2) =
K(1:2:end,1) = kvec(1) + kvec(end);
for i = 2:n-1
    K(i,i) = kvec(i) + kvec(i+1);
    K(i,i+1) = -kvec(i+1);
    K(i,i-1) =
end
```
K(n,n) = ; \% TO BE COMPLETED
K(n,n-1) = -kvec(n);
end

is intended to generate, in sparse format, the tridiagonal stiffness matrix associated with
n springs connected in series with respective spring constants kvec(i) = k_i, 1 \leq i \leq n.
Cut-paste the MATLAB code above and supply the correct conclusions to the three
assignment statements indicated by the comment TO BE COMPLETED.

(b) Write a four-line script which (i) invokes generate_K for n = 10 and kvec = ones(10,1)*10
to obtain K, (ii) finds the number of nonzero entries in K, (iii) verifies (visually) that
K is indeed tridiagonal, and finally (iv) confirms that K is stored in sparse format. You
should find the MATLAB built-in functions nnz, spy, and issparse useful.

2. The function

function [avg_time ] = timer_matvec_sparse( n, numrepeats )

kvec = ones(n,1)*n; \% spring constants for a discretized uniform truss
w = randn(n,1); \% a random displacement vector to test matrix-vector product

\% needed for Rec 12 but not Rec 11
f = ones(n,1)/n; \% a random displacement vector to test matrix-vector product

K = generate_K(n,kvec);

tic;
for itimes = 1:numrepeats
    v = K*w;
end
avg_time = toc/numrepeats;
end

computes the average time (over numrepeat repetitions) to perform the sparse matrix-vector
product K*v for the n-spring stiffness matrix K.\(^1\)

Deliverable: Write a three-line script which invokes timer_matvec_sparse (three times) to
display avg_time/n for n = 3,200, n = 6,400, and n = 12,800, and numrepeats = 100.
You should observe that avg_time/n is roughly constant and thus conclude that the time
required to perform the sparse matrix-vector product K*v increases roughly linearly with n.

3. In this question we demonstrate the advantage of sparse storage format by re-performing the
timings of Question 2 but now for K converted to (and stored in) non-sparse storage format.

(i) Create from the function timer_matvec_sparse a new function timer_matvec_full
which differs only in the introduction of one line, before you enter the for loop, which

---

\(^1\)Note the choice for kvec corresponds to a discretization of a uniform truss into n springs each associated with a
small segment of the truss of length 1/n. This discretization is not relevant to our current emphasis — computational
cost — but does ensure that the limit of large n (many springs) makes physical sense.
converts \( K \) from sparse to non-sparse/standard storage format. You should find the MATLAB built-in function \texttt{full} useful. Note that you are not changing the mathematical definition of the matrix here but rather the format in which the matrix entries are stored (or not stored) and the way in which MATLAB operates on the entries.

\( ii \) Write a three-line script which invokes \texttt{timer_matvec_full} (three times) to display \( \text{avg\_time}/(n^2) \) for \( n = 3,200, \ n = 6,400, \) and \( n = 12,800, \) and \texttt{numrepeats} = 10. You should observe that \( \text{avg\_time}/(n^2) \) is roughly constant and thus conclude that the time required to perform the non-sparse matrix-vector product \( K\*w \) increases roughly quadratically with \( n \) and that furthermore the non-sparse matrix-vector product is considerably more expensive than the sparse matrix-vector product.